フォートナイト| Epic Games

フォートナイトのワークフロー：
UnrealGameSync を使用した大規模チームでのコラボレーション

内容

はじめに

Epic Games は、2017年に *フォートナイト*を初めてリリースし、その後すぐに*『世界を救え』*、*『バトルロイヤル』*、*クリエイティブ* モードを続けてリリースしました。*フォートナイト*は3億5,000万人を超えたプレイヤーに支えられ、新しいコンテンツやゲームプレイ モード、そして一度止めてもまたやりたくなるような新機能を常に提供し続けています。

Epic Games では、これを維持するため、*フォートナイト*に加えられる変更を効果的かつ効率的に実装および提供するためのアプローチを進化させてきました。そして、このアプローチをどのように実現しているのかについて、多くの外部スタジオから質問がありました。

このドキュメントでは、開発セットアップの進歩とその背後にある理由、そして Epic Games カタログの*フォートナイト*およびその他のタイトルの実装について詳しく説明します。Epic Games で用いられている方法では、安定性を得るために、主に Unreal Editor と UnrealGameSync ツールを使用した実用的なアプローチで継続的に繰り返すことの必要性を強調しています。

ここで提供されている情報は、Unreal Editor をチームへ配布して大規模プロジェクトにおける独自の開発プロセスの合理化を検討しているテクニカルディレクター、ビルドエンジニア、テクニカルアーティスト、ゲームデザイナーに向けたものとなっています。

フォートナイト| Epic Games

フォートナイト| Epic Games

大規模なゲーム開発における課題

ゲームの開発中、プログラマー、デザイナー、アーティストは、面白く、視覚的に魅力たっぷりで、バグのないゲームを作り上げるために努力を重ねています。そして、ゲームのイテレーション開発が繰り返されている間、デベロッパーはコンテンツ クリエイター向けのビルドとツールの安定性を維持し続けなければならない一方で、最新バージョンのコードに新しいゲームプレイ機能を追加し続けなければならないという課題に直面しています。

Epic Games のデベロッパーは、*フォートナイト*のメインラインに 1 日に数百もの変更を加えていますが、インフラストラクチャとワークフローは、このような変更をできるだけスムーズに実行できるように設計されています。ただし、常にこの方法がうまくいっていた訳ではありません。

従来のワークフロー

*フォートナイト*の開発初期、コンテンツ クリエイターは Unreal Engine 3 時代に開発された *editor promotion pipeline (エディタ プロモーション パイプライン)* と呼ばれるプロセスに依存していました。

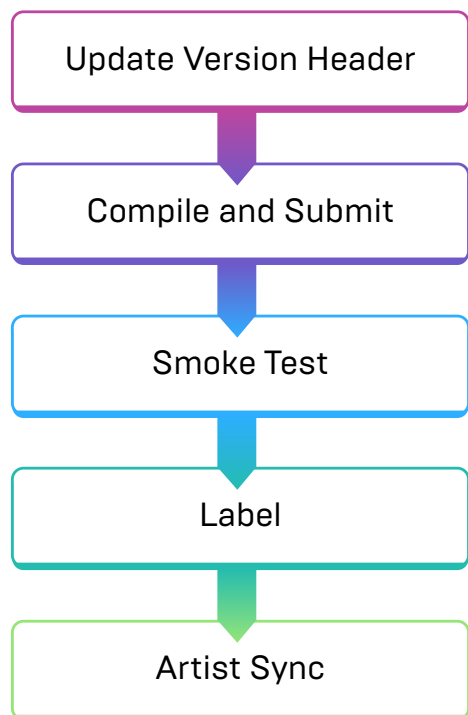



図 1：従来のエディタ プロモーション パイプライン

最新の*フォートナイト*のソースコードは、ビルドマシンで同期およびコンパイルされ、結果のバイナリは Perforce に送信されます。Epic の QA チームはそれらバイナリを同期して実行し、テスト計画に合格した場合は Perforce ラベルを適用してそれらに「良」とマークを付けます。Epic では、このプロセスを *promotion (昇格)* と呼んでいました。コンテンツ クリエイターは、Perforce ラベルを介して最新の昇格されたバイナリを同期することができました。

ビルドから修正までのサイクル

コンパイル中にエラーが見つかった場合、またはテスト中にバグが特定された場合、QA はバグをファイリングしてデベロッパーが問題を切り分けて修正を行えるようにしました。

これにより、プロジェクトの規模と複雑さが増すにつれ、テストが必要となるサーフェス領域とバグが特定される可能性も増加しました。特定のプロジェクトのアクティブなワークフローでバグがどれほど深刻な障害を与えているかを知るのに QA は適切ではなかったことから、通常、失敗しないよう十分に用心して、バグ追跡ソフトウェアで優先度の高い問題として見つかったエラーはすべて入力していました。

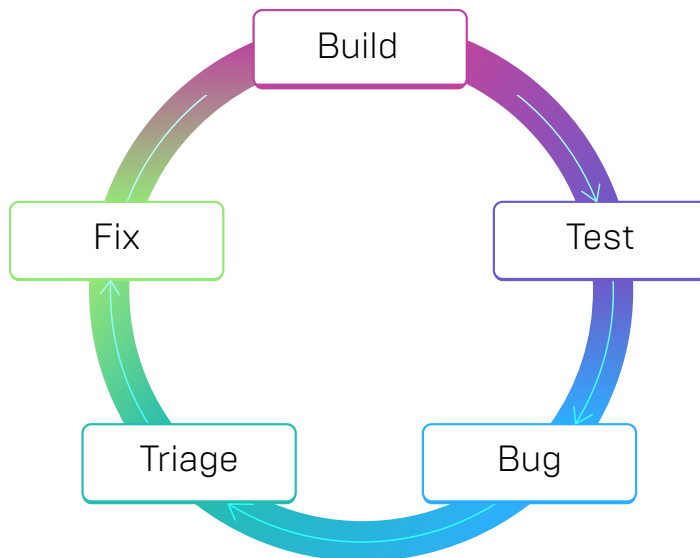



図 2：ビルドから修正までのサイクル

ビルドから修正までのサイクルの繰り返しには、最短でも半日かかります。デベロッパーが問題を再現できなかった場合や休んでいた場合は、更に数日かかることもあります。そうしてようやくプロセス全体を繰り返すことができますが、これだと 2 回目の繰り返しまでに、何かが破壊されてしまうような変更を誰かから加えられる可能性が非常に高くなってしまいます。

プロジェクトが進むにつれてターンアラウンドは遅くなり、昇進の間隔が 1 週間かかってしまうこともありました。これは、エンジニアとコンテンツ クリエイターが共同作業する上で大きな問題となっていました。

多くの Unreal Engine プロジェクトで、エンジニアとデザイナーが Unreal Engine のブループリント ビジュアル スクリプティング システムを介したゲームプレイ機能を使用して共同作業を頻繁に行っていたことから、ターンアラウンド時間の短縮は極めて重要でした。ゲーム開発にとって、繰り返しのサイクルは最重要事項ですが、これはデザイナーに新しいコードを提供しているパイプラインのボトルネックでもあります。こうしたボトルネックにより、最終プロダクトの品質をレビューして改善する時間が少なくなってしまいます。

フォートナイト| Epic Games

コンテンツ中心およびコード中心

このワークフローには他にも問題がありました。ビルド システムは常に新しいバイナリを Perforce に送信することから、エンジニアが自分でビルドしたバイナリと、同期しているバイナリの間に不一致が頻繁に発生していました。運が良かった場合、これは見つからなかった装飾シンボル名を参照する OS からの不明なダイアログボックスをトリガーすることになりますが、運が悪かった場合、エディタはロードされるものの、モジュール間で一致しないクラス レイアウトからのメモリ破損によって引き起こされる不明なコールスタックによってクラッシュを引き起こしていました。

エンジニアは、Perforce クライアントスペックからプリコンパイルされたバイナリを手動で除外することによりこの問題の回避を試みていました。しかし、このやり方がうまくいく可能性は低く、新しいファイルを導入した最新の状態の維持が困難でした。さらに、この方法は多数のワイルドカードでファイルをフィルタリングする必要があるため、Perforce サーバーに負担がかかり、同期時間が長くなることもあります。

エンジニアがコンテンツを変更する必要がある場合、コードの変更をチェックインしてエディタが昇格するのを待機した後、コンテンツの変更を送信する必要があります。ローカル ビルドで行われた変更をチェックインしようとすると、変更はバージョン管理*されず*、昇格されたエディタを使用しているデベロッパーはそれらをロードできません。

従って、エンジンをチームにデプロイするためには、バージョン管理の必要性を理解することが重要です。

バージョン管理

継続的な開発には、Unreal Engine のプロパティ シリアライゼーション モデルが柔軟性が高く便利です。このモデルでは、既存のアセットを再保存しなくても、プロパティをエディタに公開しているコードで繰り返し使用できます。

例えば、クラスにプロパティを追加し、以前に作成したアセットがそのプロパティなしで保存された場合、エンジンは次にアセットが読み込まれたときにプロパティをデフォルト値に初期化します。プロパティのデフォルト値がオーバーライドされていない場合、ディスクからロードされたものはすべて、デフォルト値に加えられた変更を自動的に取得します。また、プロパティを削除すると、そのプロパティに値を持っていたアセットは、ロード時にそのプロパティを無視します。

そして、このように明示的なバージョン管理がない場合は問題が発生する可能性があります。新しいプロパティが追加され、それとともにアセットが保存された場合、エディタはどのようにシリアル化されたのかが古いバージョンで認識されず破棄されてしまいます。そして、エディタではそのプロパティが削除されたものとして扱われます。

この問題は、ビルド変更リストをエディタの実行可能ファイルにコンパイルし作成したアセットに保存して、エディタに自身より新しいエディタで作成したものを読み込ませないようにすることで解決できます。コードの変更ごとにバージョンを細かく管理するのではなく、エディタ全体とエディタが保存するすべてのものをバージョン管理するということです。

Battle Breakers | Epic Games

ケース スタディ

*フォートナイト*の厄介なワークフローに取り組んでいる一方で、並行して *Battle Breakers* と呼ばれる独立したプロジェクトも立ち上げました。*Battle Breakers* には、ゲームを更新するための固有の高速かつ効率的なパイプラインがありました。そして、このプロセスが*フォートナイト*のようなより大きなプロジェクトに対してどのようにスケールアップできるのかを検討していました。また、別タイトルである *Paragon* がワークフローのソリューションをどのように進歩させたのかについても確認しました。

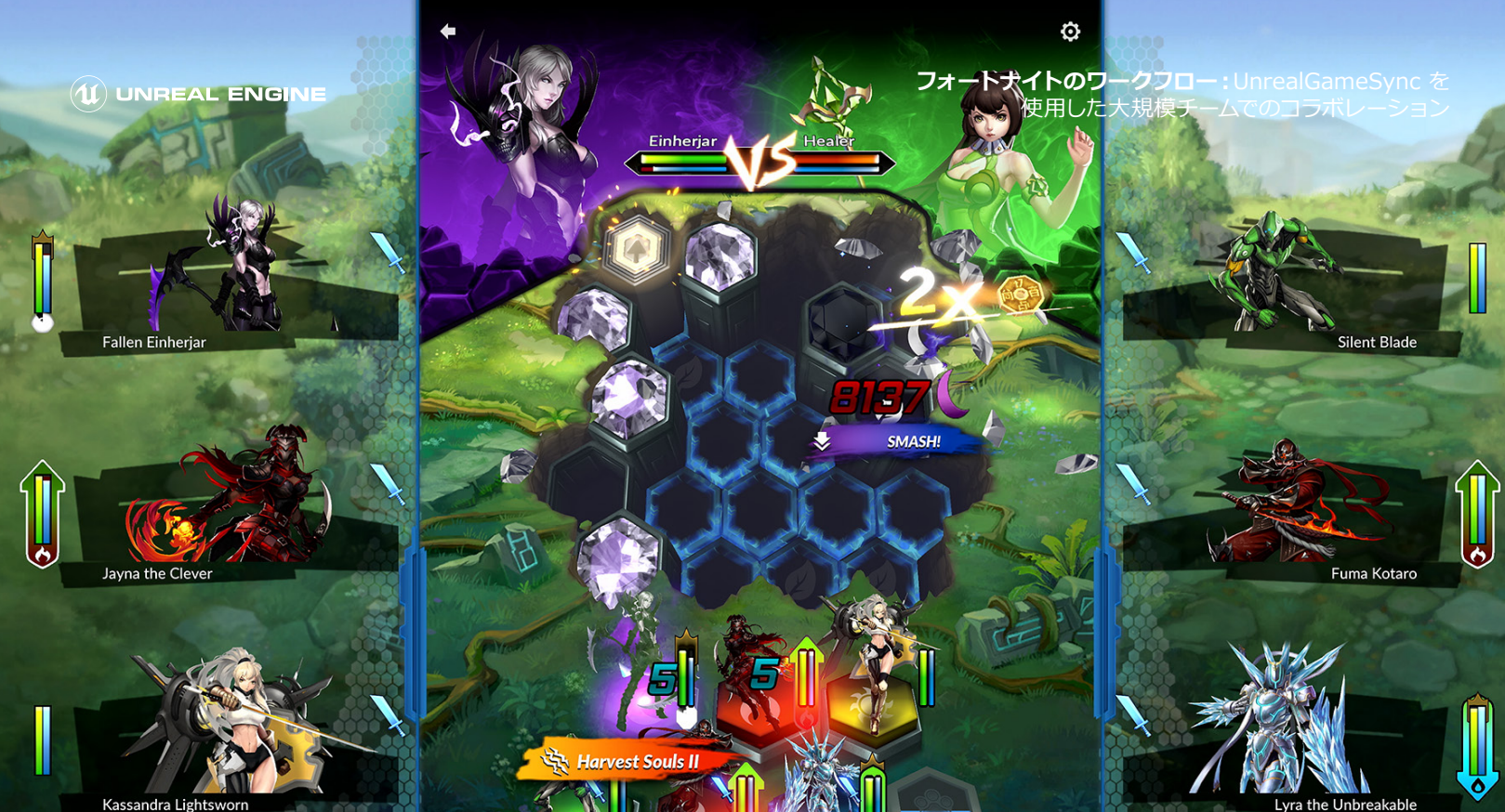
Battle Breakers

Battle Breakers は 10 人程度のチームで成り立っており、アーティスト、デザイナー、エンジニアがすべて同じ部屋で作業していました。チームメンバーは自分たちのプロジェクトに情熱を持ち、その成功に向けて力を注いでおり、極端に慎重なやり方でプロジェクトの進行が遅くなることは避けたいと考えていました。

各デベロッパー (アーティストとデザイナーを含む) は、標準のエディタ プロモーション パイプラインを採用するのではなく、Visual Studio Express のコピーを入手して、Perforce から同期し、プロジェクト ファイルを生成し、統合開発環境にソリュ ーションを読み込み、独自のエディタをビルドする方法を学びました。

このワークフローはとても効果的でした。

コンパイル エラーが発生した場合、またはエディタがクラッシュした場合は、コンテンツ デベロッパーは近くにいるエンジニアに連絡して助けを求めることができました。そして、Visual Studio は既にコンピューターにインストールされており、完全なシンボルを使用してソースコードにアクセスできたため、エンジニアはエディタを効率的にデバッグして、すばやく作業に復帰することができました。

また、アーティスト、デザイナー、エンジニア間のコミュニケーションが大幅に増加するという、思いがけない副作用もありました。それにより、メンバーはお互いのワークフローについてより深く理解し、エディタやツール、さらにはゲーム自体を改善する方法についてのアイデアを共有することができました。

Battle Breakers | Epic Games

フォートナイトにスケーリングする

Battle Breakers で行った方法を*フォートナイト*に適用してどのようにスケールアップするのかを検討した際、明確な懸念事項がいくつかありました。

安定性

まず、安定性についてです。誰もエディタの品質を評価しない場合、バグの発見はアーティストやゲームデザイナーなどのユーザー向けコンテンツを作成するメンバーによって行われることになりますが、この場合、どのような影響があるのでしょうか?

アーティストは通常、ゲームプレイ　エンジニアが頻繁に変更しない、核となるエディタのワークフロー　(メッシュ、テクスチャ、アニメーションなどをインポートするプロセス) に依存しています。ゲーム プロジェクトでは、コードの変更を行ったとしても通常はこれらのプロセスに影響を与えません。エンジンの変更は通常、個別のストリームで開発され、ゲーム　ストリームへのマージは徹底的な QA テストを行った後にのみ実行されます。

ゲーム デザイナーにとって、これはトレードオフの関係となります。つまり、生産性を上げるには安定性を犠牲にする必要があるということです。 ただし、ゲームに最も関連するコードパスを実行するのはデザイナーが最適であるという、強い主張もありました。QA は半年前のテスト計画に基づき運用されている可能性があるものの、通常、デザイナーはゲームの現在の内容の確認に関心を持ち、ゲームプレイ プログラマーの最新コードをすぐに使用したいと考えています。

データの整合性

誰かが変更をチェックインしてアセットを破壊したり、データを破損したりするのではないかという懸念がありました。ただし、事例証拠により、この種の極端なコンテンツの破損はまれであるものの、発生した場合は　QA　による簡単なテストでは事態の把握が困難であることがわかりました。

コンパイル時間

コンテンツ　クリエイターがエディタのコンパイルを待つための時間が無くなってしまうのは望ましくないものの、増分コンパイルが合理的にタイムボックス化できるめどは立ちました。ビルドが失敗するより成功する可能性が高いことを確認できた場合、休止時間中、つまり夜間または朝最初に実行するようにスケジュールできました。さらに、Epic　はコンパイル時間を大幅に短縮してチ ームへの影響を最小限に抑えるために、Xoreax の IncrediBuild を採用しました。

インターフェースの複雑さ

Visual Studio Express のようなプロフェッショナル ツールは、初心者ユーザーにとってはとっつきにくいものとなっています。インターフェースはワークフローに対して直感的ではなく、オプション配列もややこしく目の回るようなものになっています。そうしたツールを必要とするユーザーのことを考えると、インターフェースをシンプルにするのは合理的といえるでしょう。

問題をフラグする

下流工程のデベロッパーをバグから保護するためのプロセスの一部を削除する場合、問題が発生した時にその報告が簡単かつ速やかに行えることが非常に重要となります。フォートナイトの場合、チームは皆同じ部屋に集まっておらず、様々なタイムゾーンでグローバルに散らばっています。フォートナイトに採用するソリューションには、このようなタイプのチームで機能する通知システムを含める必要がありました。

解決に向けて

この種のワークフローをフォートナイトの規模のチームに展開するには、カルチャー面における劇的な変革が必要であり、妥協のない徹底的な変化を起こす必要がありました。しかし、当時使用していた既存のプロセスは遅くて扱いにくかったことから、特に失うものはありませんでした。

なので、週末に IT 部門は Visual Studio Express をフォートナイト コンテンツ クリエイターの開発用 PC にインストールし、UnrealGameSync (UGS) と呼ばれる社内で開発したカスタムツールもインストールしました。UGS は、この新しいタイプのワークフロー専用として Epic で最近開発されました。

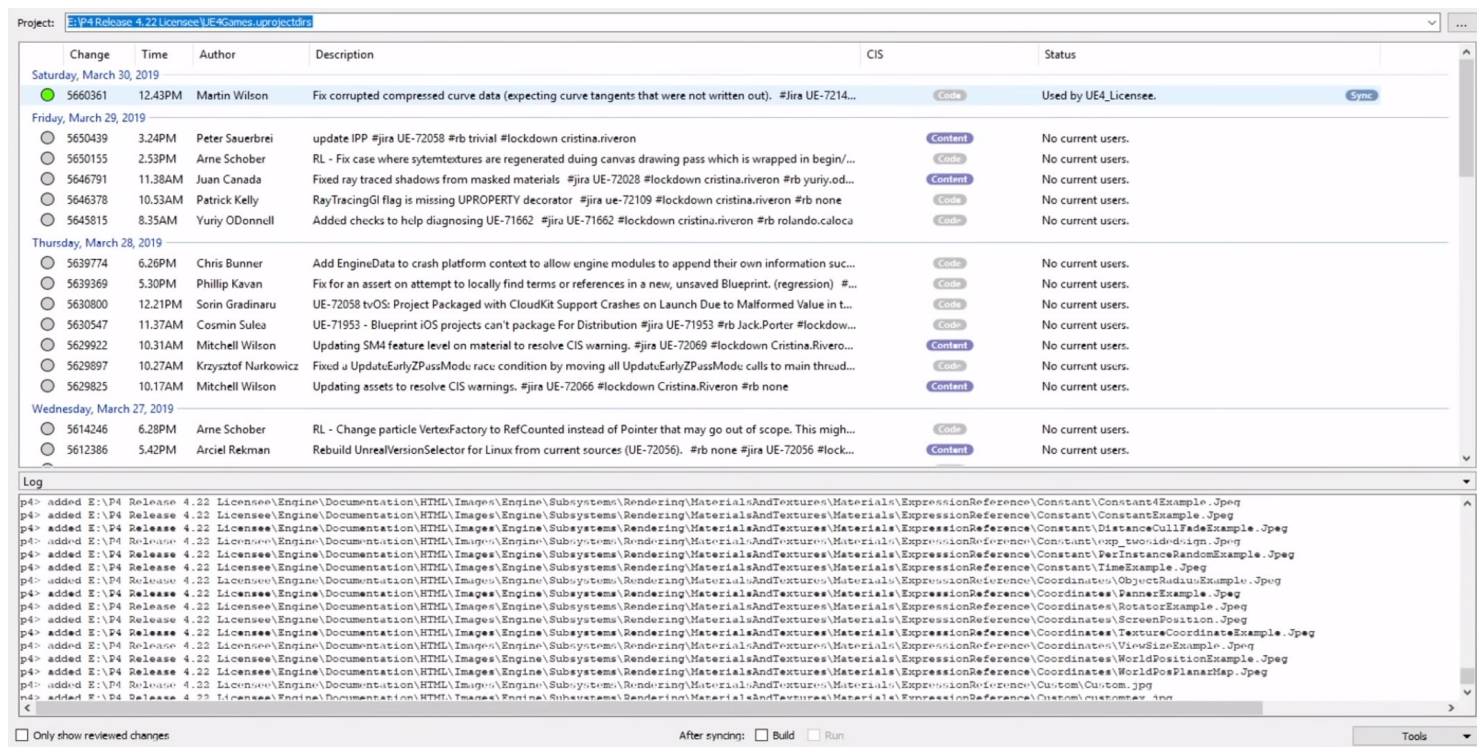



図 3：UnrealGameSync は、Perforce から UE プロジェクトを同期するためにグラフィカルなフロントエンドを提供しています。

そして、それらのプロジェクトは Microsoft *Visual Studio コンパイラでビルドすることも可能です。この画像は、UGS の初期バージョンのものです。*

概念的には、UGS はコラボレーション環境でのコードとコンテンツの統合を促進します。これは、ソースコントロールに送信された変更のライブストリームを表示します。また、変更の同期や構築、他ユーザーの行動の確認、見つかった問題へのフラグ付与を行うための合理化されたインターフェースも提供します。

コンテンツ クリエイターは、UGS 内でローカルのハードドライブからプロジェクトを選択します。UGS は、プロジェクトや使用中のワークスペースなどのPerforce 設定を自動的に検出し、そのプロジェクトに影響を与えたチェックインされている変更のリストをフェッチします。

アーティストが変更項目をダブルクリックすると、必要なファイルがローカルで同期されてコンパイルされ、その結果を含むエディタが起動します。

コラボレーション

コンパイルが成功すると、プロジェクト内の変更リストに緑色のアイコンでフラグが付けられます。このアイコンは、プロジェクトで作業している他の人から見えるようになっています。失敗した場合は、赤いアイコンが表示されます。

エディタが起動時にクラッシュしたり、その他の問題が発生したりした場合、アーティストは UGS に戻ってその変更を「bad (良くないもの)」としてマークし、何が起こったのかコメントすることができます。これにより、エンジニアリング　チームは問題の調査をどこから始めればよいかがわかります。

なので、誰かがフラグを立てた問題を調査していた時は、調査を行っているエンジニアはビルドを「under investigation (調査中)」としてマークすることができました。これにより、その後のすべての変更に対して「good (問題なし)」と再びフラグが立てられるまで、「bad」フラグが付けられていました。

ビルドをスケジュールする

リクエストされて実装された最初の機能の 1 つは、ブランチを同期し、数時間後にビルドをトリガーするユーザー設定可能なスケジュールでした。これにより、朝オフィスに来たとき、エディタが確実に最新安定版に更新された状態となっているようにしました。スケジュールで同期されるバージョンは通常、ビルドが成功したことを示す緑色のアイコンでタグ付けされたコードの最新バージ ョンとなります。

他の分野へケータリングする

エンジニアは、UnrealGameSync を使用して、エディタのビルドに使用されるバージョン管理メタデータを更新することも推奨されていました。これにより、コードを送信して昇進されたビルドを待ってコンテンツを変更するのではなく、ローカルでコンパイルされたバイナリを使用してコンテンツを変更することができるようになりました。

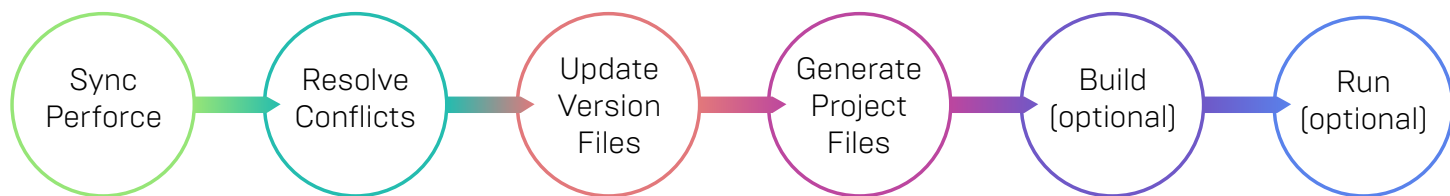



図 4：UnrealGameSync パイプライン

フォートナイト| Epic Games

フィードバックを一元化する

UGS はすぐにワークフローの定番となり、迅速なプロダクションに不可欠である素早い反復サイクルへ復帰するのにとても役立ちました。

この変更は 4 年前に行っています。フォートナイト チームは今も UnrealGameSync を使用しています。

図 5：フォートナイト チーム用にカスタマイズされた UGS (次の表のキー)

フォートナイト チームと繰り返し作業を続ける中で、他のいくつかの機能も追加されました。下の表の UGS 領域は、図 5 の番号に対応しています。

UGS エリア	**機能ノート**
1	タブが複数あるため、異なるプロジェクトまたはストリームを同時に表示できます。
2	上部のステータスパネルには、選択したプロジェクトが表示され、他のツールにアクセスできます。このパネルには、現在のブランチに必要な SDK バージョンも表示されます。 ユーザーは簡単にストリームを切り替えることができます。Perforce のファストストリーム スイッチングは、間違って実行しやすくなっており評判は良くありませんでした。なので、代わりにこのようなツールを介してこのモードを使用し公開することで、確実な実行が可能となります。
3	右側サーフェスのバッジの状態は、検証ビルドが進行中であるか、成功したか失敗したかなど、ビルドシステムに起因します。バッジをクリックすると、ユーザーはビルドログに直接移動します。 UnrealGameSync は、ブランチの変更を常にバックグラウンドでポーリングしているため、新しくチェックインされるとそれを認識します。バッジが緑から赤に変わったら、UGS は、ビルドを壊した変更リストを送信したデベロッパーに、ポップアップ通知を提供できます。

表 1：フォートナイトに追加された UGS 領域と機能

これらの機能の詳細については、 「UGS のリファレンス」を参照してください。

*フォートナイト*では、ビルド ファームで継続的なビルド サイクルを実行して、UnrealGameSync にバッジを追加します。

- 10 分ごとに、エディタを段階的にコンパイルします。各変更のサーフェス領域を削減するため、ユニティ ビルドは無効にされています。これにより、送信されたコードの欠落した **#include** ディレクティブもキャッチされます。
- エディタがコンパイルされると、コマンドレットは、前回の実行以降に変更されたコンテンツをロードします。エラーの一般的な原因は、アーティストがコンテンツの提出を忘れているということであるのがわかりました。また、エラーやそのような問題を説明する警告をすばやく生成するには、単にコンテンツをロードするのが手っ取り早いです。
- コンテンツのチェック後、エディタを 2 回起動する簡単な自動テストを実行します。1回目は **-game** 引数、2 回目は **-server** 引数を使用します。このテストでは、クライアントとサーバーが相互に通信できること、ログイン フローとマッチメイキング プロセスが正しく機能していること、およびそれらのマッチが可能なことを確認します。また、*フォートナイト*で実行できる 4 つの基本アクション、移動、射撃、建築、収集をチェックします。
- 2 台目のマシンは、すべてのターゲット プラットフォームのゲームを段階的にコンパイルします。

これらのテストを実行するこれらの 2 台のマシンは非常に価値があることが証明されており、迅速なフィードバックと幅広いカバレッジの間で適切な妥協点を取ります。

Paragon | Epic Games

Paragon

UnrealGameSync を次に試したプロジェクトは *Paragon* でした。*Paragon* は、大規模なアートチームと積極的なコンテンツパイプラインへの迅速な立ち上げが可能なマルチプレイヤーのオンラインバトルアリーナ (MOBA) でした。デザイナーは既存のブループリント機能を使用してほとんどのゲームプレイ ロジックを記述しているため、設計とエンジニアリングの繰り返しはフォートナイト の場合ほど重要ではありませんでした。

このような大規模なアートチームでは、アーティストが独自のバイナリをコンパイルするオーバーヘッドがスループットに悪影響を及ぼすと考えられていました。しかし、UGS には他にもいくつかの利点があります。エンジニアが正しくバージョン管理されたコンテンツを保存できるようにしたり、ビルドの正常性の問題に関するフィードバックを表示したり、ビルド済みのバイナリをワークスペースから除外したりできます。

圧縮されたバイナリ

ソリューションは、ビルド サーバーからのエディタ バイナリが自動的に ZIP ファイルにアーカイブされ、ブランチの外の場所に送信されるモードでした。ZIP ファイルが送信されると、コンテンツ クリエイターは 2 番目のワークスペースを管理したりローカルでビルドしたりすることなく、UGS のダウンロードと抽出を選択できるようになっていました。

これらのアーカイブは、特定の一致する変更リスト番号に対して公開され、ソースからビルドするワークフローと効率的にマッチします。マッチするプリコンパイル済みバイナリがない変更は、UI でグレー表示されます。マッチするバイナリでの変更後、コンテンツのみの変更の同期が許可されます。

プリコンパイルされたバイナリを必要としないユーザーは、必要に応じて同期してコンパイルできます。

クライアント側のフィルタリング

独自の問題を抱えていた *Paragon* では、多くのオフサイト請負業者や外注スタジオと協力しました。*Paragon* には多数の高解像度のキャラクター アセットがありましたが、通常、デベロッパーは一度に数個のアセットのみ必要でした。VPN 接続を介して作業しているすべての人にとっては、キャラクターの完全なラインナップの同期を行うと、帯域幅が浪費されてしまい、とても時間がかかる作業となっていました。

PARAGON

Paragon | Epic Games

これを解決するために最初に試みたことは、Perforce で仮想ストリームの作成でした。これにより、ユーザーのワークスペースがブランチ全体ではなくディレクトリのサブセットに絞り込まれました。これは小規模なものでは機能しましたが、順列の数は急速に増加しました。更新を絶え間なくリリースするということは、多数のブランチが発生するということを意味します。そして、コンテンツ クリエイターはすべてのブランチで個人的にカスタマイズを行いたいと考えていました。

UnrealGameSync は、Perforce サーバーでこれらの順列を管理するのではなく、クライアント側のフィルタリングをサポートします。これは、ストリームのコンテンツをサーバーに照会し、ファイルを選択的に同期 (および削除) して、ユーザーが必要なファイルのみを残します。

UI には、ユーザーが独自のフィルタを使用して補足または除外できるように、カスタマイズ可能なアラカルトのコンポーネントセットが表示されます。例えば、ユーザーはその時点で関係のないプラットフォームを除外したり、シネマティック コンテンツや特定のローカライズされたアセットを除外したりできます。

ここで選択できるコンポーネントには、古いスタイルの Perforce クライアント仕様と比べて、それぞれの構成を一元化できるという利点があります。コンポーネントを定義するコンフィギュレーション ファイルはソース コントロール自体に送信されるため、個々のユーザーによる明示的な構成変更を必要とせずに、プロジェクトの全員に対してフィルタを自動的に更新できます。

Clean Workspace (クリーン ワークスペース ツール

UE ディレクトリ構造では、プラグインとゲーム プロジェクトの中間ファイルがソースツリー全体に散在しています。

UGS の Clean Workspace ツールを使用して、中間ファイルをクリーンアップしたり、ワークスペース内にある欠落ファイルを見つけることができます。

ローカル　 ワークスペースとサーバー間のすべての差異がツリーに表示され、安全に削除できる既知の中間物がデフォルトで選択されます。つまり、作成した後にチェックインを忘れた作業が誤ってツールによって削除されることはありません。
Perforce の Clean コマンドとは異なり、Clean Workspace ツールは想定される場所でファイルがあるかどうかだけをチェ ックし、その後ファイルの長さと読み取り専用のステータスをチェックします。PC にもよりますが、それほど時間はかかりません。プロセス全体で 10～20 秒程度です。

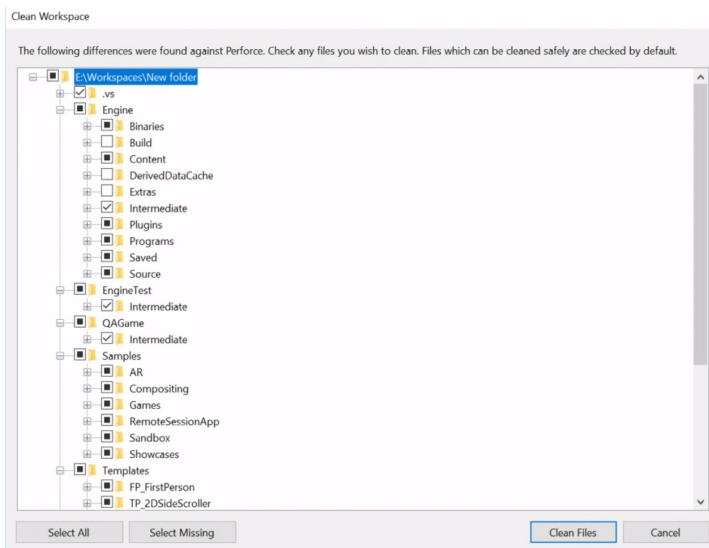



図 6：Clean Workspace ツール

Unreal Engine

UGS ワークフローの利点は、ゲームプロジェクトに対するものだけではありません。Unreal Engine の QA 部門は、リグレッションテスト時のバグに UGS も使用します。

Bisect (二等分) モード

多くの場合、QA は既知の適切な変更リストと既知の壊れた変更リストの間で機能のリグレッションが発生したことを把握しており、どれが問題の変更リストなのかをできるだけ効率的に絞り込む必要があります。UGS パネルから、テスターは変更の範囲を選択して、Bisect モードと呼ばれるものを実行できます。このモードでは、選択した変更のみが含まれるようにビューをフィルタリングします。このビューから、ユーザーは変更を良好または不良としてマークできます。

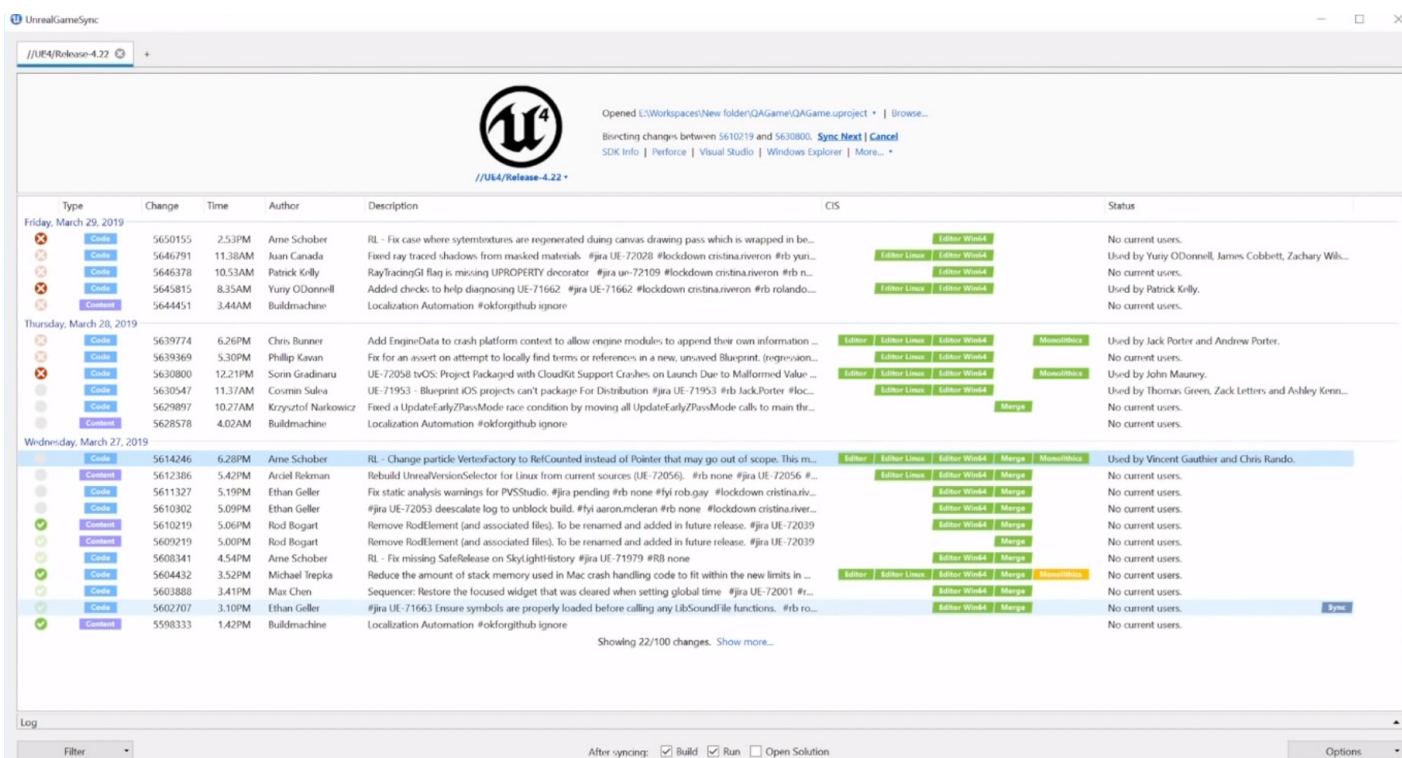



図 7 : UGS Bisect モード

テスターは、変更に同期し、テストしてすぐにバグを見つけられない場合、Sync Next コマンドを選択できます。そして、UGS は最後の適切な変更と最後の不適切な変更の中間に同期し、テスターがバグを発生させた変更のバイナリ検索を実行できるようにします。

コンフィグ ファイル

最後に、独自のプロジェクトを作る際に役立つ小さなカスタマイズをたくさん施しました。

ステータス パネルの下に、「message-of-the-day (本日のメッセージ)」機能があり、通常、ブランチする予定の日付に関する情報を提供します。message-of-the-day には、バグ データベース内のバグのリストへのリンクも含まれています。

また、提出された変更の説明にバグ番号を含める必要がある Perforce トリガーがあります。UGS を設定してその情報を解析し、バグ追跡ソフトウェアにある対応するバグへのリンクを作成します。これは、UGS で変更したものの横にあるバッジをクリックして開くことができます。

これらの機能はすべて、ブランチにチェックインされたコンフィグ ファイルに基づき、それを介して操作されます。これにより、様々なプロジェクトの固有のニーズに合わせて UGS を調整できます。

プロジェクトで UnrealGameSync を使用する

フォートナイトの成功は、チームメンバーが互いに効率よく協力し、コミュニケーションをとれるかどうかにかかっていました。Battle Breakers と *Paragon* で直面したワークフローの課題が UGS の開発につながりました。これにより、1つの部屋で共同作業する際のメリットがグローバルに分散した開発環境でも受けられやすくなりました。

Unreal Engine では最新バージョンの UGS を配布しており、どのバージョンの UE でも使用できます。そのソースコードは Perforce と GitHub で利用できるようになっており、その使用は Unreal Engine EULA でカバーされています。

UGS の設定に関する詳細情報は、Unreal Engine のドキュメントの「UGS のリファレンスページ」を参照してください。

著者

Ben Marsh

Robert Gervais

編集

Michele Bousquet

Su Falcon

レイアウトとメディア デザイン

Jung Kwak